Cost-effective Replica Management in Fault-Tolerant Cloud Environments

Bart Spinnewyn*, Juan Felipe Botero†, and Steven Latré*
*University of Antwerp - imec, IDLab, Department of Mathematics and Computer Science, Belgium
†Universidad de Antioquia, Calle 67 # 53 - 108, Medellín, Colombia

Abstract—**Cloud providers rely on fault-tolerance mechanisms to realize high-availability services on best-effort infrastructure. Service replication limits the data-loss caused by failure, at the expense of additional operational costs. Recently, with the advent of Mobile Edge Computing, cloud environments are becoming increasingly heterogeneous and dynamic, by the incorporation of (very) unreliable and resource-constrained devices. In this paper, we investigate how to devise an economically viable replication strategy, for a given service on a particular cloud environment. Previous work either focused on finding replication strategies for stateless services, ignoring recovery processes and correlated failures, or considered system dynamics, while lacking Service Level Agreement (SLA)-awareness. We approach the replica management problem as a run-time revenue maximization problem. Our proposed Dynamic Programming (DP) algorithm can generate the optimal replication strategy over the application lifetime. Through extensive simulations, we show that our algorithm significantly improves provider revenue over a wide range of cloud- and SLA-conditions, and adapt its strategy to evolving operating conditions. The results show that coupling dynamic failure models with SLA-awareness can lead to profitable replication strategies, even in cases where providers currently turn a loss.**

I. Introduction

Service availability is an important non-functional requirement, as it refers to the systems capacity to recover from failure. Availability requirements are generally subject of Service Level Agreements (SLAs) between service providers and their tenants. A typical SLA could state that a certain service must be accessible without any interruption for the next 24 hours, for the provider to receive a monetary reward, or avoid paying a penalty. Service providers realize high availability services on top of commodity hardware, through techniques of virtualization and replication. The continued growth of demand for cloud services, paired with soaring operational expenses due to increasing energy prices, require cloud providers to manage their resources more effectively [1]. Often, cloud providers replicate the services across different locations to be able to conform to the SLA requirements and cope with partially failing nodes. The main challenge is to come up with a cost-effective replication strategy that can protect against failure, while limiting expenses, based on the projected operational costs and revenue generated through SLA conformity. Operational costs are comprised of hosting and migration costs. The former considers the average monetary cost due to nodal and network resource consumption under a certain service Replication Level (RL), in the absence of any recovery or migration processes. The latter encompasses the incremental service costs during those recovery activities.

In this work we adhere to a very broad definition of services, they can be Virtual Machines (VMs) or data chunks in a replicated filesystem, such as the Hadoop Distributed File System (HDFS). We investigate the problem of managing a set of services to conform to the SLAs in terms of durability with a minimum set of resources. Durability refers to the system's ability to withstand failure in the infrastructure. For each service, a cloud manager needs to dynamically select the level of replication so that the service is guaranteed to be on-line until it is not required anymore (e.g., the end of a map reduce computational task). In this, it should cope with constantly evolving levels of failure.

Previous approaches were either based on overly simplistic failure models, or focused on analytical calculation of a metric with very limited practical meaning, namely the Mean Time To Dataloss (MTDL). In this regard, Greenan et al. go as far as referring to MTDL as the mean time to meaningless [2]. For instance, it is of very little value for a system designer to make decisions based on an MTDL exceeding 1400 years, when both the practical lifetime for an application running on this platform, and the system component are under 10 years.

Furthermore, recent trend towards geographically decentralized computing environments, such as cloud-federation, edge-, fog- and mist-computing mean that (1) computing environments are increasingly changing over time, (2) long-term failure predictions are becoming increasingly irrelevant and inaccurate and (3) failure and dynamicity are becoming more present. Therefore, static replication strategies no longer suffice, as clearly the best replication strategy will depend on a combination of system parameters. Moreover, in these decentralized systems, the mean time between failure is often much shorter than the service duration, resulting in multiple possible failures during the service's lifetime. Finally, as these environments are much more dynamic, the failure level itself also changes quite drastically over time.

Hence, intelligent and dynamic replica management algorithms are needed that can synthesize good replication strategies. In this work, we are first to introduce and analyze the Generalized Replica Management Problem (GRMP), which is the problem of finding a replication strategy that maximizes expected revenue over the service lifetime. The GRMP approaches the problem of data-loss prevention from the economical perspective of a service provider. The goal of this paper is to find an optimal placement strategy over the envisioned service lifetime. Based on the current replication state and the remaining required lifetime, the replication algorithm decides whether scaling in, scaling out, or doing nothing is most cost-effective.

We identify following major contributions. First, we introduce the GRMP. Second, we propose an exact algorithm, which considers both a dynamic replication model and SLAs to maximize provider revenue. This algorithm can be used to generate optimal policies over a wide range of operating conditions. Third, we demonstrate that our approach can significantly improve provider revenue in both time-invariant and time-variant cloud environments, as our algorithm can adapt to varying system parameters.

The remainder of this paper is structured as follows. Section II provides an overview of related work. In Section III the GRMP is introduced. We propose our solution in Section IV. Subsequently, we compare the performance of our algorithm to related works in Section V. Finally, we conclude the paper in Section VI.

II. Related work

SLAs describe the Quality of Service (QoS)-level that its users can expect. Nowadays, services can be anything ranging from VMs, containers, databases, disk images.

SLA violations can lead to hefty fines for service providers. To avoid these penalties, while relying on best-effort infrastructure (e.g., a data-center), suffering from failures caused by faulty disks, software errors, power outage and network problems, providers employ fault-tolerance. In literature, there are two main protection schemes to make applications survive failures.

First, there are schemes of protection, which before any failures have happened, pro-actively instantiate additional resources. When failures happen, there are still sufficient resources available to make sure that the service remains available. Second, reactive schemes do not instantiate any additional resources before failure occurred, but try to bring failed services back on-line by hosting them on different nodes. Reactive methods do not use any additional resources in the absence of failure. However, as bringing a new replica on-line always requires some minimal setup time, service outage cannot be avoided. Additionally, reactive methods cannot avoid that at least some data is lost. Therefore, reactive methods are typically only used to protect stateless services. In protection schemes a trade-off must be made between resource consumption and availability.

Considerable research effort has been devoted to the failure behavior in replicated systems. In a first approach, researchers have considered the failure behavior of services to be static. For instance, Jayasinghe et al. model cloud infrastructure as a tree structure with arbitrary depth [3]. Physical hosts on which VMs are hosted constitute the leaves of this tree, while the ancestors comprise regions and availability zones. The nodes at bottom level are physical hosts where VMs are hosted. Wang et al. estimate the availability of a single VM as the probability that neither the leaf itself, nor any of its ancestors fail [4]. In previous work, we have designed application-level replication techniques for heterogeneous cloud networks, where a static failure model was assumed [5]. While these approaches can serve as a first approximation to assess the hosting costs, they are not accurate as in reality failures exhibit both temporal and spatial correlation [6]. Static approaches rely on three very limiting assumptions. First, they assume a constant availability value for each machine over time. In reality, at some moment all machines seize permanently to function, consequently their instantaneous availabilities decline over time. Additionally, when a transient failure occurs it will take some minimum time before the machine is back on-line. Hence, temporal correlation in the availability of a single machine cannot be ignored. Second, these approaches typically assume the same set of machines to be used during the entire deployment. In contrast, replicated systems often migrate copies that are hosted on unrecoverable machines. These migrations will also take some time, and induce migration costs, which cannot be modeled in static models. Third, in reality a service becomes unrecoverable when insufficient copies are on-line at a given moment. Hence, the availability of a single service is time-dependent as the probability that such a catastrophic event has occurred inevitably increases over time.

Therefore, more dynamic failure models are needed to estimate the total operation cost and service availability. Google researchers propose a time-homogeneous Markov model for replicated systems [8]. They define a cell as a pool of devices, together with their higher level coordination processes. They show that replication across multiple racks, or even across geo-distributed clouds can be accurately modeled as multiple linked cells. Their model considers failure correlation, recovery and migration processes. The most important limitation of the work is that they do not consider how to generate a "good" replication strategy, given a certain system model. Their availability model forms the basis for our proposed approach.

In distributed file systems two redundancy schemes are commonly used (1) replication, which creates identical replicas for each data block, and (2) Erasure Coding (EC), which transforms original data blocks into an expanded set of encoded blocks, such that any subset with enough encoded blocks can reconstruct the original data blocks [9].

For system designers, it is important to have a replication strategy that addresses their specific challenges. The main types of resources that comprise the subjects of a cloud resource management system are compute, networking, storage and power [10]. In general, the provider's objectives related to resource consumption and the objectives of the cloud-user conflict. For instance, Silberstein et al. focus on reducing the bandwidth required for the EC recovery process [11]. The authors propose a lazy recovery scheme: recoveries are queued and bundled, hereby reducing bandwidth consumption, while also increasing the failure probability. While the authors evaluate their proposed recovery schemes for a combination of disk, machine and rack failures, they do not provide a practical algorithm which can be used to select the best strategy. In another approach, Wang et al. try to maximize reliability and at the same time keep the hosting expenses under a maximum level [12]. However, the applicability of their approach is severely limited by the assumption of a static failure model.

The objectives of the providers center around efficient and effective resource use within the constraints of SLAs with the Cloud users [10]. Therefore, the providers need replication management algorithms that consider the impact of management decisions on operational expenses and the expected payoff through SLA conformity. As these cloud environments are increasingly changing over time, the replication management algorithms must become more dynamic.

Our approach exceeds the state-of-the-art in that it focuses on the problem of revenue maximization during a certain time window. Other approaches are generally limited to finding analytical expressions for the MTDL, while lacking any notion of resource consumption costs. Moreover, our approach is not limited to an analysis of replicated systems, but yields a practically usable replication management algorithm. Finally, our proposed algorithm can generate strategies that vary over the service lifetime.

In this paper, we will focus on SLA violation as a direct result of failure. The impact of workload variations on response-time requirements is considered out-of-scope. However, we will consider the indirect consequences of workload variations on data-unavailability through their effect on system costs.

III. Generalized Replica Management Problem

In this paper, we research how one should manage the RL of a certain service over time, in order to maximize the expected revenue during its relevant lifetime. The remainder of this section is structured as follows. First, we situate the replication manager in the cloud management architecture. Next, we discuss the replication model, its parameters, and the validity of the model. Finally, we provide a description of the GRMP.

A. Cloud management architecture

A typical cloud architecture is shown in Figure 1 [10]. Service requests originate from the cloud-users. These requests are handled by the admission control function of the cloud manager. Requests can be either admitted or declined. When a request is accepted, then the replication manager decides on both the required RL and the replica placement. When a decision is made, the controller updates the placement configuration. This controller executes initial placement, recovery, migration and tear-down processes on the infrastructure. The monitoring and prediction component monitors the infrastructure and estimates the future system behavior based on monitoring data. This component models system costs, failure behavior, recovery times, and tracks the state of accepted service requests. In this paper, we will focus on the replication manager.

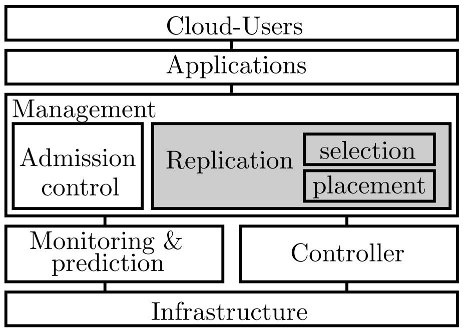


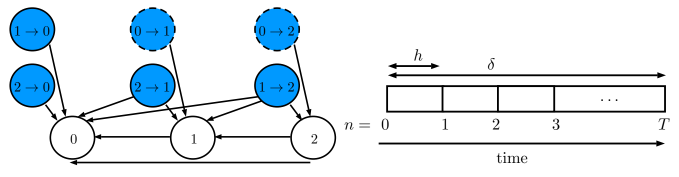
Fig. 1: Cloud architecture. This work focuses on the replication management components (grayed out).

TABLE I: Replication model parameters.

Parameter	**Description**
L_{min}	Minimum RL for the data to be recoverable.
L_{max}	Maximum RL to be considered.
$\mathbf{I_{ON}}$	States for which the data is recoverable.
$\mathbf{I_{OFF}}$	States for which the data is unrecoverable.
A_i	Set of possible actions (go-to states) from state i.
$C_{i,j}(n)$	Cost of action j in i, at the start of MI n.
$P_{i,j}(n)$	Transition probability from state i to j during MI n.
h	Management period, i.e. the time between MIs.

B. Replication model

Google researchers showed that large-scale replicated systems can be modeled as time-homogeneous Markov processes [8]. In their model, the current behavior of the replicated system is uniquely determined by the current number of copies available. Hence, the process is memory-less, i.e. as time passes the process loses the memory of the past. They represent both failure and recovery processes by constant transition rates between states. Consequently, their approach can only be used to evaluate a fixed strategy. In our model, we use a time-inhomogeneous Markov chain. We observe the process periodically and assume that the behavior can only change at the start of a new period. An overview of the replication model parameters is given in Table I. In the following we illustrate how these parameters can be determined in an HDFS context. L_{min} depends on the chosen replication scheme. When the number of active copies drops below this value, then the data is irrevocably lost. For naive replication the minimum replication level is always 1, while for (k, n)-EC, this value is equal to k. L_{max} is the maximum number of copies that can be active at the same time. For naive replication, this value is only limited by the number of nodes available in a cell. However, to limit computation time its value can be set to a reasonable upper limit, and can then be further increased should a strategy with maximum RL be chosen by our algorithm. For EC, L_{max} equals n. Figure 2a shows an example for a single-cell replicated system with $L_{min} = 1$ and $L_{max} = 2$. The states **I** in the model comprise two types. First, there are regular states (uncolored) for which the number of reserved copies, and the number of active copies are the same. There is a state corresponding to RL 0, 1, ..., L_{max}. These



(a) Illustration of a replication model. (b) Service request.

Fig. 2: Problem description.

states are also present in [8]. Second, we introduce action states (colored), which represent a decision to change the RL. In HDFS the current RL can be queried via the Application Programming Interface (API). The desired RL can be set via the same interface. The action costs can be estimated using advanced energy profiling methods, such as the one proposed in [15]. For practical purposes it typically suffices to consider the hosting costs proportional to the RL. The migration cost can be approximated as proportional to the amount of data transferred through the network. In the following, we assume that scaling out takes a certain exponentially distributed time, while scaling in happens instantaneously. For a multi-cell configuration, the states and actions can be derived in a similar way.

The action states with an intermittent border represent actions which can only be taken upon the initial deployment of the service, when 0 copies are active. The corresponding actions are assumed to take place immediately. After initial deployment, the data is available or unrecoverable, depending on whether the service state is in $\mathbf{I_{ON}}$ or $\mathbf{I_{OFF}}$ respectively. $\mathbf{I_{ON}} = \{1, 1 \to 2, 2\}$ and $\mathbf{I_{OFF}} = \{0\}$. $A_0 = \{0 \to 1, 0 \to 2\}$, $A_1 = \{1 \to 0, 1, 1 \to 2, 1 \to 3\}$, $A_2 = \{2 \to 0, 2 \to 1, 2\}$. Note that when the service is in an action state at the beginning of an MI, then the only possible action is to stay in this same state. In any state, choosing to stay in the same state, is a valid action. For any of the instantaneous actions, there are is only one possible next state. These states are not considered in $\mathbf{I_{ON}}$ and $\mathbf{I_{OFF}}$ as the service will never be observed in this state at the start of an MI.

The recovery rates can be estimated directly from historical data. The transition probabilities corresponding to failures can be determined in two ways. First, in real deployments the transition rates can be derived directly from observed replica failures, e.g., using maximum likelihood estimation. Second, they can be determined from the nodal failure distribution. Given a failure burst, we can compute the expected fraction of copies made unavailable by the burst. Assuming that copies are uniformly distributed across nodes of the cell, the replica failure rates in the model can be determined from the device failure distribution combinatorially [8]. This approach will taken in Section V.

Figure 2b illustrates that in our model the replication state of each service is observed periodically, namely at the start of each MI n. For each service the first MI ($n = 0$) takes place upon initial deployment. Subsequent MIs ($n > 0$) are separated by the management period h.

TABLE II: Service parameters as per SLA.

Parameter	Description
δ	Required lifetime in time-units.
T	Required lifetime in MIs.
R	Reward received when available at $n = T$.
V	Penalty when not available at $n = T$.

At the start of MI n, the current replication state i of the service is observed and the replication manager decides to take action $j \in A_i$. The cost incurred to the system for taking action j is given by $C_{i,j}(n)$.

C. SLA model

The following SLA is considered. A service request has a duration δ, comprising exactly T management periods. Hence, for this service a decision will be made at the start of MI $n \in \{0, 1, 2, \ldots, T-1\}$. If the service is accessible until δ time-units after initial deployment, then the service provider receives a reward R. In case the service is no longer available, then the provider receives a penalty V. Figure 2b illustrates the flow of a request. Table II provides an overview of the service parameters.

D. Formal problem description

The GRMP is formally defined as follows. *Given the replication and service model defined in Section III-B and Section III-C respectively, and that revenue of the provider is determined by the costs that he makes and the potential reward or penalty that he receives from the user. In order to maximize expected revenue, which action should the replication manager take at MIs $n = 0, 1, \ldots T-1$?*

IV. Algorithmic description

Given the inputs described in Table I and Table II, we maximize the expected reward over the entire request duration. First, the algorithmic approach is described. Then, we present the algorithm in pseudo-code.

A. Approach

The memoryless-property of the Markov model implies that the behavior of the replicated service at the start of each MI is only determined by its current replication state i, hence the same goes for the expected reward. At the start of MI n, the actions that will be taken for MI $n+1$ up-to $T-1$ are unknown. We denote the best action when in state i at MI n as $J_i(n) \in A_i$.

The expected reward of taking action $j \in A_i$ at MI n, namely $R_{i,j}(n)$, is determined by two elements. First, the immediate cost of this action. Second, the probability to transition from j at n to all other states $\tilde{j} \in \mathbf{I}$ at the start of MI $n+1$, and associated expected reward $R_{\tilde{j}}(n+1)$. Therefore $\forall n \in \{0, 1, T-1\}$:

$$R_{i,j}(n) = -C^{(i,j)} + \sum_{\tilde{j} \in \mathbf{I}} P_{j,\tilde{j}}(n) R_{\tilde{j}}(n+1) \qquad (1)$$

For state $i \in \mathbf{I}, n = T$ the reward is defined by the SLA:

$$R_i(T) = \begin{cases} -V, i \in \mathbf{I}_{OFF} \\ R, i \in \mathbf{I}_{ON} \end{cases} \tag{2}$$

Hence, using Equation 1 and Equation 2 we can recursively determine the actions that maximize expected reward. $\forall n \in \{0, 1, T-1\}, i \in \mathbf{I}:$

$$J_i(n) = \arg\max_{j \in A_i} R_{i,j}(n) \tag{3}$$

and

$$R_i(n) = \max_{j \in A_i} R_{i,j}(n) \tag{4}$$

Summarized, to maximize the overall expected reward at the start of MI n, one needs to select the action with the highest expected reward, given the current replication state i. To calculate the expected reward for each possible action, it suffices to know the cost of this action, the transition probabilities to all states in $\mathbf{I}$, and the associated expected rewards, at the start of the next MI.

B. Algorithm

Using the approach described in the previous section, we can find the best policy recursively. Algorithm 1 describes how the best strategy can be found using Dynamic Programming (DP). The values of $J_i(n)$ and $R_i(n)$ will be stored in tables $\mathbf{J}$ and $\mathbf{R}$ respectively, to prevent unnecessary recalculation.

First, the decision table $\mathbf{J}$ and the table containing the expected values $\mathbf{R}$ are initialized (Line 4 and 5). $J_i(n) == \emptyset$ signifies that the entry has not been calculated previously. `Calc(i,n)` returns the expected reward for (i, n) and writes the value in the table. All decisions and expected rewards for all states in $\mathbf{I}$ and for MI $n+1, \ldots, T$ will be either retrieved from the tables, or calculated and stored. If the decision for (i, n) was already tabulated, then the corresponding value is returned (Line 10). If after initial deployment, insufficient replicas are accessible, then the expected value is $-V$ (Line 15). When the service has reached the deadline, and sufficient copies remain, then the expected value is R (Line 20).

In all other cases, the best decision and corresponding expected reward must be evaluated for all possible actions (Line 23). Then, the action with the highest expected reward is selected (Line 25) and the maximum expected value is stored and returned (Line 27). Note that, the values are written to the table to prevent unnecessary recalculation of $\mathbf{J}$ (Lines 14, 19, 25) and $\mathbf{R}$ (Lines 13, 18, 26).

The worst-case complexity can be derived as follows. The decision table contains $|\mathbf{I}| \times (1+T)$ entries. However, for the column at $n = T$ no actions need to be calculated (Line 14 and 19). For each entry (i, n) (Line 23), $|A_i|$ possible actions must be evaluated. We introduce A as the maximum number of actions for any state. Evaluation of each action requires $|\mathbf{I}|$ multiplications and additions. Hence, filling in the table is $\mathcal{O}(|\mathbf{I}|^2 TA)$. Note that we opted for a recursive formulation to ease readability. In a real deployment, overhead should be limited by an iterative implementation.

Algorithm 1 Proposed optimal replication algorithm.

```
 1: var $\mathbf{I}, \mathbf{I_{ON}}, \mathbf{A}, \mathbf{C}, \mathbf{P}, T, R, V$
 2: for each $i \in \mathbf{I}$ do
 3:     for each $n \in 0, 1, \ldots T$ do
 4:         $J_i(n) = \emptyset$
 5:         $R_i(n) = 0$
 6:     end for
 7: end for
 8: procedure CALC($i, n$)
 9:     if $J_i(n)\ ! = \emptyset$ then
10:         return $R_i(n)$
11:     end if
12:     if $i \notin \mathbf{I_{ON}}$ && $n > 0$ then
13:         $R_i(n) = -V$
14:         $J_i(n) = 0$
15:         return $-V$
16:     end if
17:     if $N == T$ && $i \in \mathbf{I_{ON}}$ then
18:         $R_i(n) = R$
19:         $J_i(n) = 0$
20:         return $R$
21:     end if
22:     for each $j \in A_i$ do
23:         $R_{i,j}(n) = -C^{(i,j)} + \sum_{\tilde{j} \in \mathbf{I}} P_{j,\tilde{j}}(n)\textsc{Calc}(\tilde{j}, n+1)$
24:     end for
25:     $J_i(n) = \arg\max_{j \in A_i} R_{i,j}(n)$
26:     $R_i(n) = R_{i,J_i(n)}(n)$
27:     return $R_i(n)$
28: end procedure
```

V. PERFORMANCE EVALUATION

We implemented a discrete event simulator that adheres to the architecture depicted in Figure 1. The simulator was implemented in Java. We simulate a single cell with 100 nodes. Since the model approximates devices within a single cluster to be homogeneous, individual nodal and bandwidth requirements can be ignored, as long as the aggregate capacity within the cluster suffices. Additionally, the number of nodes does not impact the expected reward, as only the nodes on which the service are hosted impact the costs and availability. However, increasing the simulated node count does impact the simulation time as more failure events will be generated. Failures arrive at a rate λ. For the number of failures in one failure-event we used the bi-exponential model proposed by Nath et al [16]. The probability of a failure event where i out of u nodes fail is given by

$$p_i = (1-\alpha) f(\rho_1, i) + \alpha f(\rho_2, i) \tag{5}$$

For each node the Mean Time To Failure (MTTF) is given by

$$MTTF = \frac{u}{\lambda \sum_{i=1}^{u} (i p_i)}. \tag{6}$$

Equation 6 can be used to generate failures with the same burst size distribution, but with different MTTFs by varying λ. We consider $\alpha = 0.009$, $\rho = 0.3$ and $\rho = 0.96$, unless specified otherwise. These values were extracted from real live deployments on PlanetLab [16]. In the experiments, recoveries can only be initiated at the start of a MI. To prevent accumulation of failure events during an MI, the period must be sufficiently smaller than the MTTF. Under this condition, the exact value of the MI does not influence the expected reward.

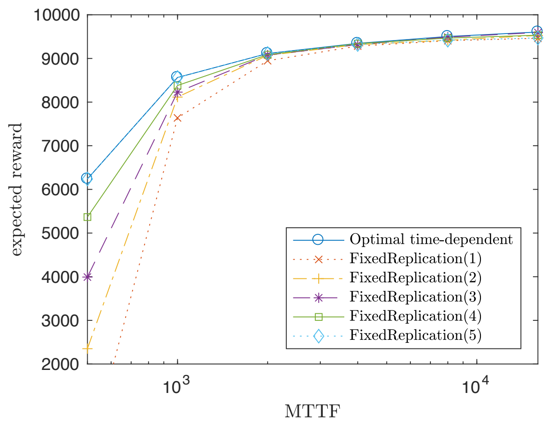


Fig. 3: Static naive replication: influence of the MTTF on expected reward.

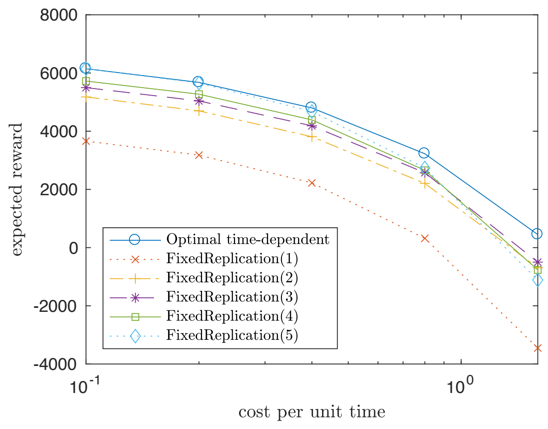


Fig. 4: Static naive replication: influence of the hosting cost on expected reward for MTTF=8000.

A. Static naive replication

1) Simulation setup: In this setup, we consider a replicated system with L_{min} and L_{max} equal to 1 and 5 respectively. A hosting cost model with a cost per unit time of 0.1 per replica is used. The cost is proportional to the number of duplicates reserved. Two replication algorithms are considered. First, *Optimal time-dependent* is our proposed algorithm. Second, *Fixed Replication(L)* is the default replication strategy in HDFS. This strategy always tries to return to a predefined RL, $L = 1, 2, 3, 4, 5$.

2) Results: The influence of the MTTF on the expected reward is shown in Figure 3. The request duration is 1000, and $h = 100$. R and V are 1000 and 10000 respectively. The mean instantiation delay equals 10, for any recovery process. For all MTTFs the expected reward per request of our proposed algorithm is equal to, or higher than for the fixed replication strategies. For low MTTF values the expected reward is dominated by the impact of SLA violation. The expected reward per request is equal to that for $FixedReplication(5)$. When the MTTF goes up, then the expected reward increases for all fixed RLs. For high MTTFs failures are very unlikely and the best policy will be to minimize the operational costs. In Figure 4, the cost per unit time is varied from 0.1 to 16. The hosting cost has a dramatic effect on the expected reward. Clearly, our algorithm performs best over all cost levels as it is aware of the operational costs. For low operating costs, the maximum RL is optimal. However, when the cost increases, then the maximum RL becomes the worst. For a cost per unit time of 0.8 our algorithm performs 17% better then the others. For a cost per unit time of 1.6 our approach is still profitable (+448) while the others have a negative expected reward (-498 and worse). These results show the importance of considering operational costs.

Next, we investigate the influence of request duration on the expected reward in Figure 5. $MTTF = 8000$, instantiation delay=10, $h = 100$, cost per unit time=0.1, R=10000, V=100000. Reward and penalty are assumed proportional to the request duration, which is more or less realistic. For low request durations, all six algorithms perform similarly as the probability of failure is negligibly low. When the request duration increases, then the expected reward increases for all algorithms. The performance of $Fixedreplication(5)$ coincides with our algorithm. Note that this is because the cost per unit time is only 0.1. Hence, for the maximum duration of 20000 time-units, the total operating cost per request can only vary between 0 and 1000. However, the pay-off can be either 10000 or -100000.

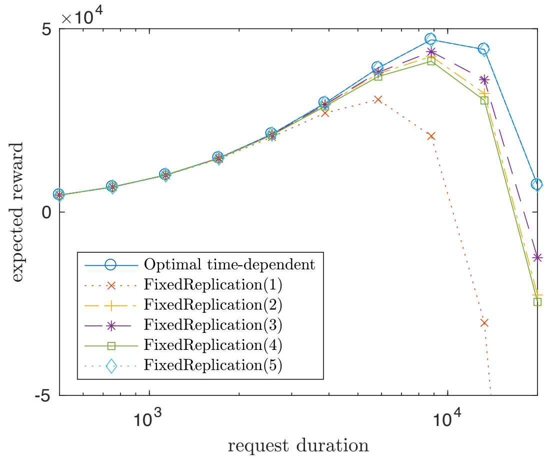


Fig. 5: Static naive replication: influence of the required lifetime on the expected reward.

The associated computation time to generate the decision Table for each request is shown in Figure 6. The computation time is averaged out over 10 runs. The computation time goes up linearly with the number of decision intervals, as determined in Section IV-B.

B. Static Erasure Coding replication

1) Simulation setup: In this setup we consider a $(10, 14)$-erasure coded system. This time as cost we consider the data transferred by the recovery process. To completely recover from f failures, with a block-size b, $(f + k) \times b$ must be transferred. We compare our algorithm to $LazyReplicaton(L)$, for

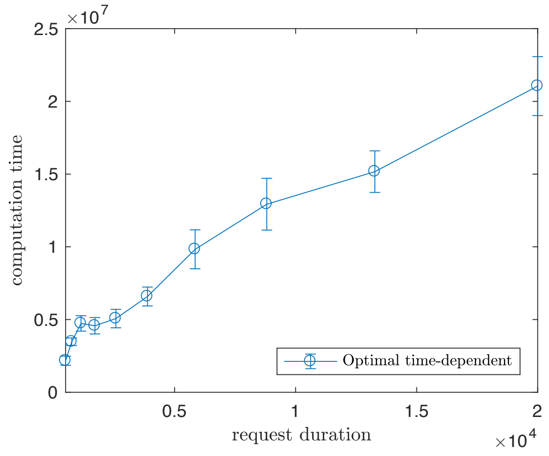


Fig. 6: Static naive replication: influence of the required lifetime on computation time.

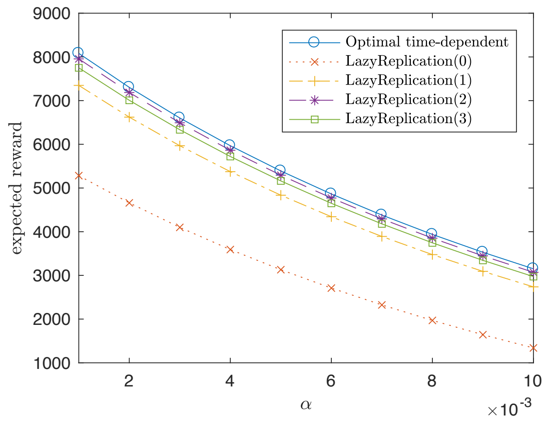


Fig. 7: Static EC replication: expected reward as a function of the failure correlation (stripesize=256MB).

$L = 0, 1, 2, 3$ [11]. A recovery is only instantiated when more than L failures have been detected.

Again the request duration is 1000. $R = 10000$, $V = 100000$, $h = 100$, $MTTF = 8000$. The mean recovery time is 50.

2) Results: In Figure 7 and Figure 8 the probability of a large scale failure burst α is varied. For low α the failures are relatively uncorrelated, while for high α there is a high probability of a large-scale failure burst.

Clearly, the expected reward goes down as the failure correlation increases. Overall, our proposed algorithm exceeds the performance of the others. Note that for a stripesize of 256MB $LazyReplication(2)$ performs second best, while for a stripesize of 512 MB $LazyReplication(3)$ performs second best. This can be explained by a doubling of the recovery cost, which results in an optimal policy with a higher probability of data-loss.

Figure 9 and Figure 10 show the relation between data-loss probability and expected transfer costs for the generated policies. The reward is varied logarithmically between 10^2 and 10^8, and the penalty equals 0. It is clear that the performance

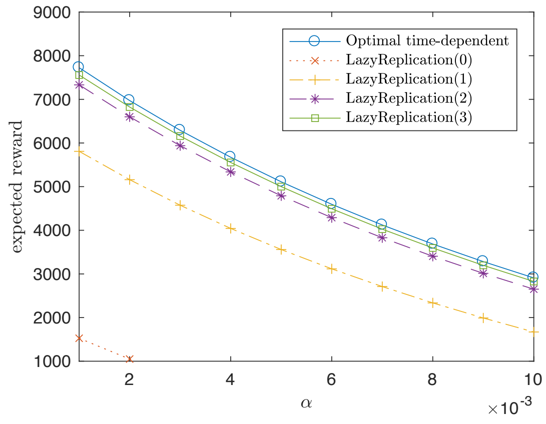


Fig. 8: Static EC replication: expected reward as a function of the failure correlation (stripesize=512MB).

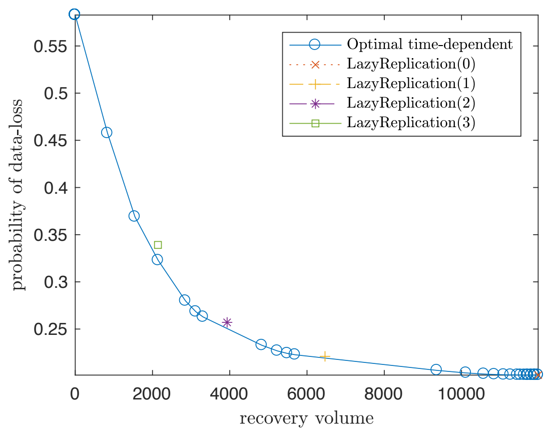


Fig. 9: Static EC replication: relation between recovery volume and probability of data-loss (MTTF=2000).

of our proposed algorithm dominates the Lazy Replication algorithm. Moreover, the distribution of data-points shows that the proposed algorithm can generate a rich set of replication strategies, that trade-off recovery costs for data-loss. This figure illustrates the importance of balancing availability and operational costs to maximize expected revenue.

C. Dynamic replication

1) Simulation setup: In this setup we consider naive replication with minimum 1 and at most 10 replicas. Again, we consider a hosting cost per unit time of 0.1. We simulate a dynamic environment, where the failure behavior changes abruptly. $MTTF = 5000$ for $t \in [0, 500000[$ and equals 10000 for $t \in [500000, +\infty[$.

The request duration is increased to 10000, with $h = 100$, with both R and V equal to 100000. The mean time between requests is exponentially distributed with mean 100. In total 10000 service requests are generated.

Every 100000 time units the failure arrival rate λ is re-estimated using a rolling window of 100000 time units. Based on this $\hat{\lambda}$ the failure probabilities are re-estimated and a new decision table is generated. Because the reward and duration

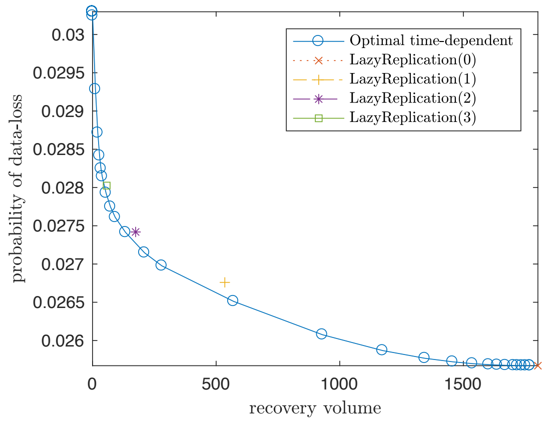


Fig. 10: Static EC replication: relation between recovery volume and probability of data-loss (MTTF=15000).

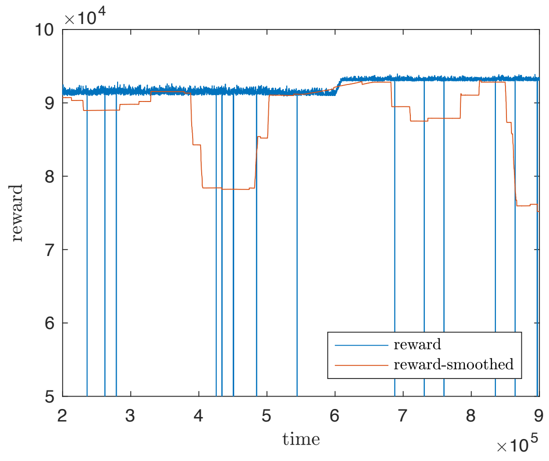


Fig. 11: Dynamic replication: reward per request.

of the generated requests are identical, this table can be shared by all services.

2) Results: For each service request the accumulated reward (including penalty, reward and costs) and the accumulated cost is tracked. At the start of each MI the current

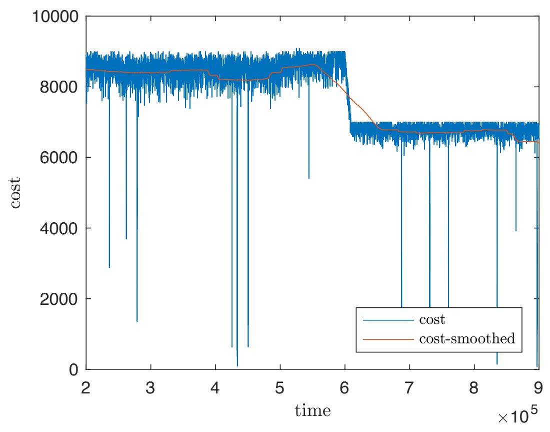


Fig. 12: Dynamic replication: cost per request.

replication state is evaluated. When the service is off-line, then a penalty is incurred. In case the deadline has been reached, then a reward is added. In both cases a reward or penalty is added and the accumulated reward is written to a logfile. No future MIs are scheduled for this service. In case the deadline was not reached (yet), then the replication manager consults the decision table. If the decision table is older than 100000 time units, then a new decision table is generated. The replication manager selects the next action from the table and the corresponding cost is accumulated.

Figure 11 and Figure 12 show the received rewards and hosting costs incurred for the generated requests. The smoothed traces result from a moving average with window size 1000. While the failure probability is lowered at $t = 500000$, the decision table is only updated with the new value of $\hat{\lambda}$ at $t = 600000$. In $[500000; 600000[$ the expected reward increases gradually, as the probability of data-loss goes down. While this is not immediately clear in the raw data, the smoothed line clearly goes up. In this same period the average costs increase slightly, because more services live through their entire required lifetime, resulting in a higher overall resource consumption. At $t = 600000$ the decision table is updated. The reward goes up linearly in $[600000; 610000[$, because the decision table is updated for running services. At $t = 610000$ we see that both average reward and cost remain stable. Compared to $t < 500000$ our replication algorithm has switched to a lower RL, increasing the average reward and lowering operating costs.

VI. Conclusion and Future Work

In this work, we present a novel unified approach towards the analysis or replicated systems, and the synthesis of a cost-effective replication strategy. We approach the problem of replica management in fault-tolerant cloud environments as a run-time revenue optimization problem. While some related works have either solely focused on analysis of replicated systems, others made very unrealistic assumptions on recovery and failure rates. Our approach is based on a dynamic failure model and can be applied to both single datacenters and geo-distributed cloud environments. As the cell dimensions go up, the underlying model becomes more accurate and the computation time remains the same. Therefore, our approach is particularly of interest for revenue optimization in very large-scale cloud networks.

We show that, while other works propose fixed placement strategies for specific cloud environments, our adaptive algorithm yields significant profitability improvements in a wide range of cloud and SLA conditions.

The major limitation of our work is that the computation time of the proposed algorithm is proportional to the number of MIs. Currently, the MI must be small to the MTTF. Therefore, future work includes switching between replication strategies, instead of deciding on individual replication operations at the start of an MI. This modification relaxes the requirement on the MI, in that it only has to be small compared to the required service lifetime.

References

[1] B. Bermejo, C. Guerrero, I. Lera, and C. Juiz, "Cloud resource management to improve energy efficiency based on local nodes optimizations," *Procedia Computer Science*, vol. 83, pp. 878 – 885, 2016, the 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050916302125

[2] K. M. Greenan, J. S. Plank, and J. J. Wylie, "Mean time to meaningless: Mttdl, markov models, and storage system reliability," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Storage and File Systems*, ser. HotStorage'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 5–5. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863122.1863127

[3] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement," in *Services Computing (SCC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 72–79.

[4] W. Wang, H. Chen, and X. Chen, "An availability-aware virtual machine placement approach for dynamic scaling of cloud applications," in *Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on*. IEEE, 2012, pp. 509–516.

[5] B. Spinnewyn, R. Mennes, J. F. Botero, and S. Latré, "Resilient application placement for geo-distributed cloud networks," *Journal of Network and Computer Applications*, vol. 85, pp. 14–31, 2017.

[6] S. Ghiasvand, F. M. Ciorba, R. Tschüter, and W. E. Nagel, "Lessons learned from spatial and temporal correlation of node failures in high performance computers," in *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on*. IEEE, 2016, pp. 377–381.

[7] S. Ramabhadran and J. Pasquale, "Analysis of long-running replicated systems."

[8] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems." in *OSDI*, 2010, pp. 61–74.

[9] R. Li, Y. Hu, and P. P. Lee, "Enabling efficient and reliable transition from replication to erasure coding for clustered file systems," *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[10] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.

[11] M. Silberstein, L. Ganesh, Y. Wang, L. Alvisi, and M. Dahlin, "Lazy means smart: Reducing repair bandwidth costs in erasure-coded distributed storage," in *Proceedings of International Conference on Systems and Storage*. ACM, 2014, pp. 1–7.

[12] T. Wang, X. Lu, and M. Hou, "Novel algorithm for distributed replicas management based on dynamic programming," *Journal of Systems Engineering and Electronics*, vol. 17, no. 3, pp. 669–672, 2006.

[13] P. Karmakar and K. Gopinath, "Are markov models effective for storage reliability modelling?" *arXiv preprint arXiv:1503.07931*, 2015.

[14] L. F. Bertuccelli and J. P. How, "Estimation of non-stationary markov chain transition models," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE, 2008, pp. 55–60.

[15] F. Chen, J.-G. Schneider, Y. Yang, J. Grundy, and Q. He, "An energy consumption model and analysis tool for cloud computing environments," in *Proceedings of the First International Workshop on Green and Sustainable Software*. IEEE Press, 2012, pp. 45–50.

[16] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Subtleties in tolerating correlated failures in wide-area storage systems." 2006.